Biztonság és védelem az informatikában

6. gyakorlati feladat

Készítette: Baranyi Gábor
CRC7FC

2021.03.24.

Puffer túlcsordulásos támadási módok

Stack alatt gyakran egy konkrét megvalósítását értünk, amely a legtöbb számítógép-architektúrán központi szerepet tölt be a programok futásában.

Ebben az értelemben a stack (vagy stacktár) a számítógép memóriájának egy része, amelybe más adatok mellett a processzor azokat a memóriacímeket menti el, ahova az egyes eljárások befejeztével visszatér.

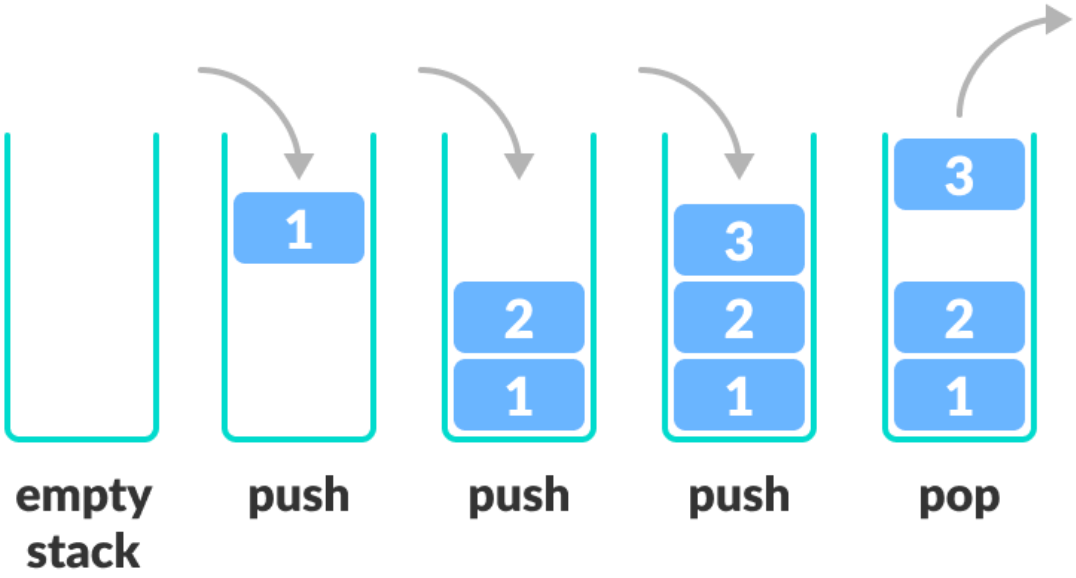
A stack-nek a szubrutinok használatakor van jelentősége: ide lehet menteni a szubrutin kezdetén azokat a regisztereket, melyek értéke meg fog változni a szubrutin végrehajtása során, valamint lokális adattárolásra is lehetőséget ad. A paraméterek átadása is általában a vermen történik.

Elhelyezkedése, működési elve:

A stack fizikailag a memóriában helyezkedik el. Dinamikus elem, mérete hossza és helye változó. A stack implementálásához egy speciális regiszterben elhelyezkedő mutató szükséges, mely mutatót (a regiszter tartalmát) speciálisan a stacket kezelő utasítások mozgatják, változtatják, szükség szerint.

A stackben többnyire regiszterek tartalmát tároljuk átmenetileg. Ennek oka az, hogy a mikroprocesszor leggyorsabban a belső regiszterekkel tud műveleteket végezni. A regiszterek száma viszont korlátozott. Például gyakran előfordul, hogy az összes regiszter már olyan információt tartalmaz, amely még nem felülírható, de az adott részfeladat elvégzéséhez szükség van további regiszterek használatához. Ekkor valamely regiszter vagy regiszterek tartalmát ideiglenesen a stackbe tudjuk kivinni, majd később a stackből a regiszter tartalmát vissza tudjuk állítani és a regiszterbe már aktuálisabb tartalmat tudunk betölteni. Ez a művelet általában gyorsabb és kényelmesebb, mint a memóriába írni a regisztertartalmat. Hiszen ilyenkor meg kellene választani a címzést, meg kellene jegyezni a tárolási címet és a tárolt adat hosszát.

Régebbi mikroprocesszoroknál úgynevezett belső stack létezett, vagyis a processzoron belül volt a stack, ami jelentősen korlátozta a processzor kapacitását. Ma minden processzornál a RAM-ban elhelyezhető 'külső stack' található.



Puffer túlcsordulási támadási módok:

1. Heap túlcsordulás:

A Stack a folyamat memóriájának egy olyan régiója, amelyet dinamikus változók tárolására használnak. Ezek a változók a malloc () és a calloc () függvényekkel vannak kiosztva, és átméretezésre kerülnek a realloc () függvény használatával, amelyek a C beépített függvényei. Ezek a változók globálisan elérhetők, és ha a memóriát kiosztjuk a kupacon, akkor felelősségünk, hogy felszabadítsuk ezt a memóriaterületet használat után. Két helyzet okozhat stack túlcsordulást

Ha folyamatosan kiosztjuk a memóriát, és használat után nem szabadítjuk fel a memóriaterületet, az memóriaszivárgást okozhat - a memória továbbra is használatos, de más folyamatokhoz nem áll rendelkezésre.

```c
// C program to demonstrate heap overflow
// by continuously allocating memory
#include<stdio.h>

int main()
{
    for (int i=0; i<10000000; i++)
    {
       // Allocating memory without freeing it
       int *ptr = (int *)malloc(sizeof(int));
    }
}
```

Ha dinamikusan nagyszámú változót rendelünk hozzá.

```c
// C program to demonstrate heap overflow
// by allocating large memory
#include<stdio.h>

int main()
{
    int *ptr = (int *)malloc(sizeof(int)*10000000));
}
```

2. Stack túlcsordulás:

A Stack a folyamat memóriájának egy speciális régiója, amelyet a függvényen belül használt helyi változók, a függvényen átadott paraméterek és visszatérési címeik tárolására használnak. Amikor egy új helyi változót deklarálunk, a stackre toljuk. A függvény befejezése után az összes függvényhez tartozó változó törlődik, és felszabadul az általuk használt memória. A felhasználónak nincs szüksége a stackterület manuális felszabadítására. A stack a Last-in-First-Out adatstruktúra.

Számítógépünk memóriájában a stack mérete korlátozott. Ha egy program a memória méreténél több memóriaterületet használ, akkor a stack túlcsordulása következik be, és a program összeomlásához vezethet. Két esetben fordulhat elő stack túlcsordulás:

Ha nagyszámú helyi változót deklarálunk, vagy tömböt vagy mátrixot deklarálunk, vagy bármely nagyobb méretű nagyobb dimenziós tömb túlcsordulást eredményezhet a stackben.

```c
// C program to demonstrate stack overflow
// by allocating a large local memory
#include<stdio.h>

int main() {

   // Creating a matrix of size 10^5 x 10^5
   // which may result in stack overflow.
   int mat[100000][100000];
}
```

Ha a függvény rekurzívan végtelen időnek hívja magát, akkor a stack nem képes tárolni az összes függvényhívás által használt sok helyi változót, és a stack túlcsordulását eredményezi.

```c
// C program to demonstrate stack overflow
// by creating a non-terminating recursive
// function.
#include<stdio.h>

void fun(int x)
{
    if (x == 1)
       return;
    x = 6;
    fun(x);
}

int main()
{
   int x = 5;
   fun(x);
}
```

Megelőzési módszerek:

A címtér véletlenszerűsítése (ASLR) - véletlenszerűen mozog az adatrégiók címterének helyein. A puffertúlcsordulási támadásoknak általában ismerniük kell a futtatható kód helyét, és a címterek véletlenszerűsítése ezt gyakorlatilag lehetetlenné teszi.

Adatfuttatás megakadályozása - a memória bizonyos területeit nem futtathatóként vagy futtathatóként jelöli meg, ami megakadályozza a támadás futtatását a nem futtatható régióban.

Strukturált kivételkezelő felülírási védelem (SEHOP) - segít megakadályozni a rosszindulatú kódok támadását a strukturált kivételkezelés (SEH), a hardver- és szoftverkivételek kezelésére szolgáló beépített rendszer ellen. Így megakadályozza, hogy a támadó képes legyen használni az SEH felülírási kizsákmányolási technikáját. Funkcionális szinten az SEH felülírást stack alapú puffer túlcsordulás alkalmazásával érik el, hogy felülírják a szál stack-ben tárolt kivétel regisztrációs rekordot.

Operációs rendszerek védekezési módjai:

A puffertúlcsordulás elleni védelmet a leggyakoribb puffertúlcsordulások észlelésére használják annak ellenőrzésével, hogy a stack nem változott-e vissza egy funkció visszatérésekor. Ha megváltoztatták, a program szegmentálási hibával lép ki. Három ilyen rendszer a Libsafe, a StackGuard és a ProPolice gcc javítás.

A Microsoft által végrehajtott Data Execution Prevention (DEP) mód kifejezetten védi a strukturált kivételkezelő (SEH) mutatóját a felülírástól.

Erősebb stackvédelem lehetséges a stack kettéválasztásával: egy az adatokhoz és egy a függvény visszatéréséhez. Ez a felosztás jelen van a negyedik nyelven, bár nem biztonsági alapú tervezési döntés volt. Ettől függetlenül ez nem teljes megoldás a puffertúlcsordulásra, mivel a visszaküldési címen kívüli érzékeny adatokat továbbra is felül lehet írni.

Források:

https://en.wikipedia.org/wiki/Buffer_overflow

https://hu.wikipedia.org/wiki/Verem_(adatszerkezet)

https://www.geeksforgeeks.org/heap-overflow-stack-overflow/?ref=rp